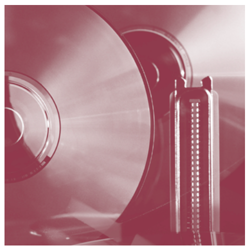
ASG-DesignManager™

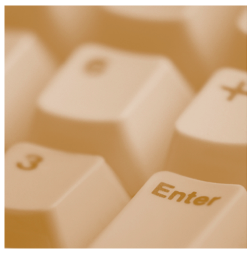
IMS (DL/I) Database Design - First Cut Modeling

Version: 1.4.3

Publication Number: DSR0200-143-DLI

Publication Date: July 1989





ASG Worldwide Headquarters Naples, Florida USA | asg.com
1333 Third Avenue South, Naples, Florida 34102 USA Tel: 941.435.2200 Fax: 941.263.3692 Toll Free: 1.800.932.5536

ASG Documentation/Product Enhancement Fax Form

Please FAX comments regarding ASG products and/or documentation to (941) 263-3692.

Company Name	Telephone Number	Site ID	Contact name

Product Name/Publication	Version #	Publication Date
Product:		
Publication:		
Tape VOLSER:		

Enhancement Request:

ASG Support Numbers

ASG provides support throughout the world to resolve questions or problems regarding installation, operation, or use of our products. We provide all levels of support during normal business hours and emergency support during non-business hours. To expedite response time, please follow these procedures.

Please have this information ready:

- Product name, version number, and release number
- List of any fixes currently applied
- Any alphanumeric error codes or messages written precisely or displayed
- A description of the specific steps that immediately preceded the problem
- The severity code (ASG Support uses an escalated severity system to prioritize service to our clients. The severity codes and their meanings are listed below.)

If You Receive a Voice Mail Message:

1. Follow the instructions to report a production-down or critical problem.
2. Leave a detailed message including your name and phone number. A Support representative will be paged and will return your call as soon as possible.
3. Please have the information described above ready for when you are contacted by the Support representative.

Severity Codes and Expected Support Response Times

Severity	Meaning	Expected Support Response Time
1	Production down, critical situation	Within 30 minutes
2	Major component of product disabled	Within 2 hours
3	Problem with the product, but customer has work-around solution	Within 4 hours
4	"How-to" questions and enhancement requests	Within 4 hours

ASG provides software products that run in a number of third-party vendor environments. Support for all non-ASG products is the responsibility of the respective vendor. In the event a vendor discontinues support for a hardware and/or software product, ASG cannot be held responsible for problems arising from the use of that unsupported version.

Business Hours Support

Your Location	Phone	Fax	E-mail
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775	941.263.2883	support@asg.com
Australia	61.2.9460.0411	61.2.9460.0280	support.au@asg.com
England	44.1727.736305	44.1727.812018	support.uk@asg.com
France	33.141.028590	33.141.028589	support.fr@asg.com
Germany	49.89.45716.300	49.89.45716.400	support.de@asg.com
Singapore	65.224.3080	65.224.8516	support.sg@asg.com
All other countries:	1.941.435.2201		support@asg.com

Non-Business Hours - Emergency Support

Your Location	Phone	Your Location	Phone
United States and Canada	800.354.3578 1.941.435.2201 **Secondary Numbers:** 800.227.7774 800.525.7775 **Fax:** 941.263.2883		
Asia	011.65.224.3080	**Japan/Telecom**	0041.800.9932.5536
Australia	0011.800.9932.5536	**New Zealand**	00.800.9932.5536
Denmark	00.800.9932.5536	**South Korea**	001.800.9932.5536
France	00.800.9932.5536	**Sweden/Telia**	009.800.9932.5536
Germany	00.800.9932.5536	**Switzerland**	00.800.9932.5536
Hong Kong	001.800.9932.5536	**Thailand**	001.800.9932.5536
Ireland	00.800.9932.5536	**United Kingdom**	00.800.9932.5536
Israel/Bezeq	014.800.9932.5536		
Japan/IDC	0061.800.9932.5536	**All other countries**	1.941.435.2201

ASG Web Site

Visit http://www.asg.com, ASG’s World Wide Web site.

Submit all product and documentation suggestions to ASG’s product management team at http://www.asg.com/products/suggestions.asp

If you do not have access to the web, FAX your suggestions to product management at (941) 263-3692. Please include your name, company, work phone, e-mail ID, and the name of the ASG product you are using. For documentation suggestions include the publication number located on the publication’s front cover.

Contents

Preface

This *ASG-DesignManager IMS (DL/I) Database Design - First Cut Modeling* provides information about designing a first cut (DL/I) database with ASG-DesignManager (herein called DesignManager). DesignManager is the interactive data and enterprise modeling system for logical and physical database design.

Allen Systems Group, Inc. (ASG) provides professional support to resolve any questions or concerns regarding the installation or use of any ASG product. Telephone technical support is available around the world, 24 hours a day, 7 days a week.

ASG welcomes your comments, as a preferred or prospective customer, on this publication or on any ASG product.

About this Publication

This publication consists of these chapters:

- Chapter 1, "Introduction to DL/I First Cut Database Design," provides an introduction to the generation of an IMS (DL/I) first cut database design from the network representation of a conceptual schema.
- Chapter 2, "Producing DL/I Command Output," describes the output produced from the three commands provided to help you to design an IMS (DL/I) database.
- Chapter 3, "Guide for the DL/I Database Designer; Use of DL/I Command Output," is a step-by-step guide to the database design process using the output from the commands provided by this facility.

Publication Conventions

Allen Systems Group, Inc. uses these conventions in technical publications:

Convention	Represents
ALL CAPITALS	Directory, path, file, dataset, member, database, program, command, and parameter names.
Initial Capitals on Each Word	Window, field, field group, check box, button, panel (or screen), option names, and names of keys. A plus sign (+) is inserted for key combinations (e.g., Alt+Tab).
`lowercase italic monospace`	Information that you provide according to your particular situation. For example, you would replace *`filename`* with the actual name of the file.
`Monospace`	Characters you must type exactly as they are shown. Code, JCL, file listings, or command/statement syntax. Also used for denoting brief examples in a paragraph.
Vertical Separator Bar (|) with underline	Options available with the default value underlined (e.g., Y|<u>N</u>).

Notation For Statement Formats

The notation used for the specification of statements (for commands and member definition statements) is the same as that used in InfoBank for the branches from which this publication was produced.

Syntax	Meaning
`KEYWORDS`	The keyboard, or a truncated version of the keyboard, must appear as specified in the syntax. Keywords are in upper-case letters and highlighted.
`variables`	Variables are substituted by some other element, as specified in the rules for the command or member type. Variables are in lower-case letters and are not highlighted.
`...`	The preceding element can be repeated.
`<A>`	A is optional.

`<B>`
`<C>`
	You can optionally include either B or C. (Note that both pairs of brackets must be aligned).

`<D/E>`	You can optionally include either D or E.
`<,FFF>...`	You can optionally include ,FFF one or more times.

Syntax	Meaning

`(G)`
`(H)`
	You must include either G or H. (Note that both pairs of brackets must be aligned).

`(I/J)`	You must include either I or J.
`(,KKK)...`	You must include , KKK one or more times.
`n, n1, n2...`	Unsigned integers.

If the syntax will not fit unto one line, the overspill is right justified, for example:

```
<DIRECTLY>(USES/CONSTITUTES) member-name <form><VIA clause>
                                                 <DIRECTLY>
```

Alternative spellings are not shown.

Strings that do not obey the rules for member names need to be in delimiters. Delimiters can be single quotes ’ or double quotes " but the opening and closing delimiters must be the same character. Your Systems Administrator may have specified an additional character that may be used as an alternative delimiter character. You can find out if this has been done by entering the command:

```
QUERY STRING-DELIMITER
```

1 Introduction to DL/I First Cut Database Design

Once you have generated the network representation of the conceptual schema, you can use the optional facility, IMS (DL/I) Database Design - First Cut Model (selectable unit DSR-PHO5), to convert the network schema to a first cut (DL/I) database design.

There are two main stages in the design of a DL/I database. The first stage is the design of the database. It includes the collection of data elements into segments and the collection of segments into hierarchies, with DL/I logical relationships and secondary indexes specified where appropriate. The second stage is concerned with the physical storage of data, including topics such as access methods, device types, block lengths, groupings, insert/replace/delete rules, and pointer options.

This facility is concerned with the first stage, the logical design of the DL/I database. It is assumed that the user of this facility has knowledge of and understands the basic concepts of DL/I.

Due the complexity of DL/I, it is generally not possible to convert the network schema directly to a DL/I database design in a single step. Using both experience and knowledge, the DL/I database designer must make the final design decisions, taking into consideration the structural design alternatives (which usually occur with DL/I) and performance parameters such as access frequencies, required response times, and volumes. However a great many of the logical processes involved in DL/I database design can be automated.

The optional facility described here provides a comprehensive set of reporting and diagramming capabilities which automate the major portion of the effort in designing a DL/I database. The facility converts the records of a network schema into DL/I segments. It identifies those segments which are DL/I root segments and uses each as the starting point of a distinct DL/I physical hierarchy. Logical relationships and access paths are displayed, and secondary key access paths are reported. In addition, the facility assists you by indicating the database design decisions you must make and what alternatives are available to you in making them.

DL/I maps and reports are provided which can be used to accomplish these tasks:

- List DL/I root segments
- Map DL/I hierarchies
- Display access paths
- Show logical relationships
- Report secondary key access paths
- Report segments.

In particular, the maps highlight DL/I design inconsistencies and other circumstances requiring special DL/I design decisions.

The description of this facility which follows contains detailed discussions of the DL/I commands, their output, the design alternatives they indicate, and the step-by-step procedures to be followed by database designers when using DesignManager to generate a first cut model (the logical design) of a DL/I database.

If you also have the optional Load Factor Calculation facility (selectable unit DSR-PH10) installed, you can use the Load Factor Analysis Report to calculate required access frequencies and response times.

ASG recommends that you read Chapter 3, "Guide for the DL/I Database Designer; Use of DL/I Command Output," on page 13 in conjunction with real output from an example of your own choosing. You do this by generating a conceptual schema (input your userviews and/or entities, MERGE them into the Workbench Design Area, DESIGN the conceptual schema, LIST the records of the network representation, and NAME some or all of the records) and then issuing the DL/I LIST, MAP, and REPORT commands.

2 Producing DL/I Command Output

Overview of the DL/I Command Output

The DesignManager optional DL/I facility provides you with reports and displays which help you design a DL/I database. They are produced by DesignManager DL/I commands, as follows:

- DL/I LIST produces a list of the DL/I root segments identified by DesignManager.
- DL/I MAP produces diagrams of one of the following (depending on the keyword entered in the command):
  - — The physical hierarchies of DL/I segments generated in the Workbench Design Area (WBDA) by DesignManager,
  - — The unidirectional logical relationships identified by DesignManager (you can yourself define additional relationships for the design),
  - — The access paths derived by DesignManager from input userviews and entities.
- DL/I REPORT produces a report either:
  - — On the secondary key access paths derived by DesignManager,
  - — On all the DL/I segments present in the WBDA.

If you issue any one of these commands (subsequent to generating a network schema in the Workbench Design Area), every record of the network schema is used to generate a DL/I segment with the same name (if one has been assigned) and number. The following sections describe the action of the DL/I commands more fully, including detailed descriptions of the output produced.

The DL/I List Command Output

To obtain a list of DL/I root segments, enter this command:

```
DL/I LIST ROOTS < SEGMENTS > ;
```

The action of the command is the same whether or not you specify the keyword SEGMENTS.

A *root segment* is defined to be a segment that has no physical parent. Each root segment is used as the starting segment of a separate physical hierarchy.

For each root segment listed, the output includes the name (if one has been assigned) and its number in the Workbench Design Area (identical to the name and number of the record from which the segment was generated).

The DL/I Map Command Output

General Description of the DL/I MAP Command Output

The common characteristics of the various types of diagrams are described under these headings:

- Layout of the output
- Use of pointers
- Use of directories

Layout of the DL/I MAP Command Output

In each diagram output by the DL/I MAP command, the segments are displayed as *boxes* and *pointers*, and the relationships (physical or logical) linking them are displayed as connecting lines. A segment can be displayed in a diagram only once as a box. Elsewhere it must appear as a pointer to the box.

A pointer is displayed in the same shape as a box but with a differing outline. In particular:

- A box is displayed with a dashed outline and a pointer with a dotted outline.
- The interior of each box and pointer contains the name of the segment it represents or, if the segment is unnamed, the interior is left blank.
- The segment number appears on the lower left boundary.

Segment names and numbers are given by the records from which the segments have been generated.

Each diagram displayed in the DL/I MAP command output begins with a *starting segment* which appears as a box in the upper left-hand corner of a new page. The segments of a diagram (represented by the boxes and pointers) and their connecting links are laid out on the output medium in *logical lines* (or rows). Each logical line occupies six physical print lines and contains one or more boxes and at most one pointer. The logical lines appearing in the output of each DL/I MAP command are numbered consecutively from diagram to diagram. The starting segment of each diagram appears by itself (as a box) on the first logical line of the diagram. Logical line numbers are useful in helping you to locate any segment from any diagram of the DL/I MAP output quickly. They are the reference numbers that appear both in the pointers (on the right upper boundary) and in the reference directories located at the end of the output. The use of pointers and the use of directories, respectively, are discussed in the next two panels. In each diagram, the boxes and pointers are laid out on the logical lines from left to right in a tree- like structure. They are placed on the logical lines in column levels, where each box or pointer at any given level is connected to only a single box at the preceding level and to no other box or pointer at the given level. A simple example of such a structure is given below (no pointers appear in the example).

```
Logical Line
    number      Level 1   Level 2        Level 3
              +---------+
1             | A       |
              +-<3>-----+
                    |
2                   |   +---------+      +---------+
                    |---| B       |--|---| C       |
                    |   +-<4>-----+  |   +-<5>-----+
3                   |                |
                    |                |   +---------+
                    |                +---| D       |
                    |                    +-<6>-----+
                    |   +---------+
4                   +---| E       |
                        +-<7>-----+
```

In the preceding example, the segment numbers are taken from the corresponding record numbers. Recall that the first record need not be assigned the number 1.

Use of Pointers in the DL/I MAP Command Output

Segments may be displayed as pointers instead of boxes for any of several reasons. They are used to highlight at a glance any DL/I inconsistencies encountered by DesignManager or any other circumstances requiring special DL/I design decisions (depending on the type of DL/I MAP command being executed).

For example, suppose that DesignManager finds that:

- More than one segment has the same physical child (a DL/I inconsistency),
- More than one segment has the same logical parent,
- More than one segment requires access to a particular destination segment.

Then, in any of these circumstances, instead of displaying the common segment repeatedly as a box along with any lower level segments it may be connected to, DesignManager displays the segment only once as a box. Elsewhere it is displayed as a pointer to the logical line containing the box. The number of this logical line appears on the right upper boundary of the pointer.

Any lower level segments connected to the segment appear only with the box, not with the pointers. Consider the sample diagram below.

```
Logical Line
    number     Level 1   Level 2        Level 3          Level 4
            +---------+
1           | A       |
            +-<4>-----+
                  |
2                 |   +---------+      +---------+
                  |---| B       |--|---| C       |
                  |   +-<5>-----+  |   +-<6>-----+
3                 |                |
                  |                |   .......(4).
                  |                +---: D       :
                  |                    :.<7>.....:
                  |   +---------+      .......(2).
4                 +---| D       |--|---: C       :
                      +-<7>-----+  |   :.<6>.....:
                                      |   +---------+     +---------+
5                                     +---| E       |-----| F       |
                                          +-<8>-----+     +-<9>-----+
```

In the preceding diagram, note that:

- Segment D appears as a pointer at level 3 because it is displayed as a box at level 2.
- Segment `C` appears as a pointer at level 3 on logical line 4 because it was already displayed as a box at that level on a preceding logical line (line 2).
- No pointers are required for segments E and F because they are lower level segments connected to segment D and their repetition is indicated by the pointer for segment D.

In the example, the segment numbers are taken from the corresponding record numbers.

There is another important use of pointers. A pointer is required when there is not enough space on a logical line to complete the path of segments appearing on it. That is, the segment being processed has one or more lower level segments connected to it, but there is not enough space on the line to display any of them.

In this case, the segment being processed is displayed at the end of the logical line as an *off-page pointer* rather than as a box. Then, at the end of the diagram, the incomplete path is continued (as a continuation diagram) on a new page. The continuation diagram starts with the same segment displayed as a box in the top left-hand corner of the page on a new logical line (just as if it were the starting segment of a new diagram).

It is the number of this logical line that appears on the right upper boundary of the corresponding off-page pointer. Similarly, the number of the logical line from which the path is continued appears on the right upper boundary of the box representing the continuation segment at the start of the continuation diagram. There is no other circumstance in which a box has an entry on its right upper boundary.

Thus, you can always tell when a box displayed at the top of a new page is the continuation segment for an incomplete path or the starting segment of a new diagram. A continuation segment has a logical line number on its right upper boundary, whereas a starting segment does not.

Use of Directories in the DL/I MAP Command Output

At the end of the complete DL/I MAP output, you will find two directories, the *Numeric Directory* and the *Alphabetic Directory*. These can be used to locate a particular segment in the output, both as a box and (where applicable) as a pointer.

The Numeric Directory is ordered by segment number. For each segment displayed in the output, it shows the following:

- In the first column (headed SEGMENT), the segment number and its name (if a name has been assigned)
- In the second column (beaded LINE), the logical line number on which the segment is displayed as a box
- In the third column (variously headed, depending upon the type of output specified in the DL/I MAP command), every logical line number in which the segment is displayed as a pointer.

The Alphabetic Directory contains exactly the same information, but only for the segments which have been named. They are listed in alphanumeric order of segment name.

As indicated previously, pointers are used to highlight DL/I inconsistencies or other circumstances requiring special design decisions. The third column in both directories is used to indicate such pointers.

For example, in the output from the DL/I MAP PHYSICAL HIERARCHY command, the third column is headed OTHER PHYSICAL PARENTS. This serves to highlight a segment which is a physical child of more than one parent segment. When this is the case, column two contains a reference to the logical line on which it is represented as a physical child by a box. Column three contains references to every logical line on which it is represented as a physical child by a pointer.

Alternatively, the third column entry for a segment can be a logical line reference to a single off-page pointer. You can distinguish between a third column entry referring to a pointer and a third column entry referring to an off-page pointer as follows. If the reference is to an off-page pointer, then the box for the continuation segment (given as a logical line reference in column two) is displayed in the top left hand corner of a new page. As indicated in the preceding panel, this box has, on its right upper boundary, the number of the logical line from which the incomplete path is continued.

The third column also provides another way to distinguish between a continuation segment and the starting segment of a new diagram. A continuation segment has a single logical line number entered in the third column of the directory, whereas a starting segment has no entry at all in the third column.

An example of a Numeric Directory is given below. It is the directory that would be output immediately after the diagram illustrating the use of pointers which was shown previously.

```
                         NUMERIC DIRECTORY
                         -----------------
   SEGMENT            LINE     OTHER PHYSICAL PARENTS
--------------------------------------------------------
 4 A                    1
 5 B                    2
 6 C                    2               4
 7 D                    4               3
 8 E                    5
 9 F                    5
```

The third column heading shown in the example, OTHER PHYSICAL PARENTS, would appear only if PHYSICAL HIERARCHY had been specified in the DL/I MAP command. The headings would differ as follows for the other variants of the command:

- If UNIDIRECTIONAL-RELATIONSHIPS were specified, the heading would be OTHER LOGICAL PARENTS
- If ACCESS-PATHS were specified, the heading would be OTHER ACCESS PATHS.

The Individual DL/I MAP Commands

The DL/I MAP PHYSICAL HIERARCHY Command

The output produced by this command represents the physical parent-child relationships that exist between the generated segments. For each DL/I root segment derived by DesignManager, a diagram is displayed showing the hierarchical structure of its dependent segments, each new physical hierarchy beginning on a new page.

To produce one or more parent-child diagrams, enter this command:

```
DL/I MAP < PHYSICAL > HIERARCHY;
```

The action of the command is the same whether or not the keyword PHYSICAL is specified.

The starting segment of each physical hierarchy is always a root segment. The root segment appears in the upper left-hand corner of a new page. A separate hierarchy is produced for each root segment, whether or not it has any physically dependent segments. A root segment which has no physical dependents is shown as a separate single-segment hierarchy and is a candidate for a *root-only database*.

In the DL/I MAP PHYSICAL HIERARCHY output, as in the case of all the DL/I MAP commands, the segments of each diagram are represented by boxes and pointers. They are displayed, along with their connecting links, along consecutively numbered logical lines. For each segment displayed, the Numeric and the Alphabetic Directories at the end of the output give the logical line(s) on which it appears as a box and (if applicable) as a pointer.

In the diagrams produced by the DL/I MAP PHYSICAL HIERARCHY command, a connecting link is displayed only between a parent segment and its child segment. It appears as a connecting line rather than an arrow, in contrast to the links produced by the alternative forms of the DL/I MAP command.

The segments, as displayed along the logical lines, represent hierarchical levels. The highest hierarchical level is the root segment, which appears in the upper left-hand corner of a new page. Successively lower levels are represented further and further to the right on a logical line. Child segments having the same physical parent are displayed one below another, all at the same hierarchical level. Each is connected to the parent by a line. Logical lines are numbered consecutively throughout each hierarchy and from hierarchy to hierarchy.

The DL/I MAP PHYSICAL HIERARCHY output is similar to that of the IBM DL/I mapping utilities (IMS MAP/VS 5796-PCY, IMS MAP 5796-PCW and DL/I MAP 5796-PCB). However, the DesignManager output is effectively turned on its side when compared with the IBM map output. That is, a child segment is displayed to the right of its physical parent rather than below it, and child segments with the same physical parent are displayed above or below one another rather than side by side.

Pointers are used to highlight segments which are physical children of more than one physical parent segment. If a segment has more than one physical parent, it is represented by a box just once. Thereafter, it is displayed as a pointer to the logical line containing the box. These pointers can be located by means of the directories which appear at the end of the output.

The DL/I MAP UNIDIRECTIONAL-RELATIONSHIPS Command

The output produced by this command represents the inherent *unidirectional logical relationships* that DesignManager has identified amongst the generated segments. To produce one or more diagrams of these relationships, enter this command:

```
DL/I MAP UNIDIRECTIONAL-RELATIONSHIPS;
```

Each relationship is from one segment, the logical child segment, to another, the logical parent segment, where the former contains the concatenated key of the latter. In the diagrams, each relationship is displayed as a single-headed arrow directed from the logical child segment to the logical parent segment.

It is also possible for a unidirectional logical relationship to exist from a logical parent segment to its logical child segment because the parent, in turn, contains the concatenated key of the logical child. In this case, the logical relationship is displayed as a bidirectional arrow, single-headed in both directions. It should be interpreted as a bidirectional rather than a unidirectional logical relationship and can be handled by methods detailed in a subsequent series of panels on DL/I database design decisions.

As in the case of all the DL/I MAP commands, the segments are represented in the diagrams by boxes and pointers laid out with their connecting links along consecutively numbered logical lines. For each segment displayed, the Numeric and the Alphabetic Directories at the end of the output give the logical line(s) on which it appears as a box and (if applicable) as a pointer.

Each new diagram of logical relationships begins with a starting segment appearing in the upper left-hand corner of a new page. If a segment is a logical child in two or more logical relationships, then its corresponding logical parents are displayed one below the other, with a single-headed arrow pointing from the logical child segment to each of the logical parent segments.

In the event of a logical child segment pointing to a logical parent segment which in turn is a logical child segment pointing to another logical parent segment in a second logical relationship, and so on, they are displayed as a path of logical relationships along the same logical line. Each successive relationship in the path appears to the right of the relationship that precedes it.

Each diagram produced consists of one or more paths of relationships. Each path contains two or more connected segments.

Pointers highlight the circumstance of a logical parent segment being pointed to by more than one logical child segment. If two or more logical child segments point to the same logical parent segment, then only the first occurrence of the logical parent segment is displayed as a box. Each subsequent occurrence of the logical parent segment (each with a different logical child segment pointing to it) is displayed as a pointer to the logical line containing the box. These pointers can be located by means of the directories which appear at the end of the output.

The DL/I MAP ACCESS-PATHS Command

The output produced by this command represents the *primary key access paths* between segments that have been derived by DesignManager from the input userviews and entities.

To produce one or more diagrams of these access paths, enter this command:

```
DL/I MAP ACCESS-PATHS;
```

Each access path is derived either from a multivalued dependency defined in a USERVIEW member of the Modeling Dictionary or from a MULTI-ATTRIBUTES or MULTI-ASSOCIATION clause of an ENTITY member.

It provides access from the key of one segment to the key of another segment (the destination segment). As such, it is referred to here as a primary key access path. *Secondary key access paths*, on the other hand, provide access from non-key sets of data elements.

A non-key set of data elements is defined as for relations and records.

In an output diagram, access between two segments is depicted by a double-headed arrow directed from one segment to the other. When access is required in both directions between two segments, the segments are shown connected by a bidirectional arrow, double-headed in each direction. That is, each is a destination segment of the other.

As in the case of all the DL/I MAP commands, the segments are represented in the diagrams by boxes and pointers laid out with their connecting links along consecutively numbered logical lines. For each segment displayed, the Numeric and the Alphabetic Directories at the end of the output give the logical line(s) on which it appears as a box and (if applicable) as a pointer.

Each new diagram of access paths begins with a starting segment appearing in the upper left-hand corner of the page. If access paths are required from one segment to several other segments, then the destination segments are displayed one beneath the other, with a double-headed arrow pointing from the source segment to each of the destination segments.

If an access path is required from one segment to another, which in turn requires an access path to yet another segment, and so on, then they are displayed consecutively as a connected sequence of access paths along the same logical line, each successive path appearing further and further to the right. Each diagram produced consists of one or more such connected sequences of access paths.

Pointers highlight the circumstance in which two or more segments require access to the same destination segment. In this case, only the first occurrence of the destination segment is displayed as a box. Each subsequent occurrence of the destination segment is displayed as a pointer, with the number on the right upper boundary indicating the logical line which contains the box. These pointers can be located by means of the directories which appear at the end of the output.

The DL/I Report Command Output

The DL/I REPORT SECONDARY-ACCESS-PATHS Command

To produce a report of the secondary key access paths between segments which are derived by DesignManager from input userviews and entities, enter this command:

```
    DL/I REPORT SECONDARY-ACCESS-PATHS;
```

Each access path is derived from a multivalued dependency specified in a USERVIEW member of the Modeling Dictionary.

A *secondary key access path* indicates that access is required from a *non-key* set of data elements in a source segment to the *key* of a target segment, where the source segment itself can also be the target segment. It is also possible for more than one secondary key access path to be defined from a given source segment to the same target segment (in each case from a different non-key set of data elements in the source segment).

For each secondary key access path derived by DesignManager, the report output includes a separate line containing this information:

- The number and, if one has been assigned, the name of the source segment.
- The number and, if one has been assigned, the name of the target segment.
- The name of each data element in the non-key set of data elements comprising the secondary search key in the source segment.

A secondary key access path can be defined to DL/I as a secondary index only if the source segment either is a physically dependent segment of the target segment or is itself the target segment. Otherwise, the access path must be defined to DL/I as a logical relationship.

The DL/I REPORT SEGMENTS Command

To produce a report of all the segments generated, enter this command:

```
DL/I REPORT SEGMENTS;
```

For each segment reported, the output includes the segment name (if one has been assigned), its number in the Workbench Design Area, its concatenated key, and its non-key fields.

You can use the command output to determine the sequence key of each segment reported. See "Identifying Sequence Keys for the Segments" on page 25.

3 Guide for the DL/I Database Designer; Use of DL/I Command Output

Overview of Logical DL/I Database Design Processes

Once you are satisfied that you have a sound and stable network schema at the conceptual level, DesignManager provides major assistance in converting the schema to a DL/I database design.

There are a number of logical processes that must be performed in designing a DL/I database. DesignManager fully automates most of these processes and provides decision support for the remainder.

This Chapter provides a detailed discussion of these processes and shows you how to use the DL/I command output to identity the design decisions you must make.

ASG recommends that you read this chapter in conjunction with real output generated from an example of your own. The diagrams shown in this chapter are not presented in the form of DL/I command output. They are intended to illustrate the design decisions you can encounter when using the DL/I command output to help design a database.

Following is a summary of the steps you would perform in generating a logical DL/I database design (in each case, indication is given of any DesignManager command that can be used in performing the step):

1. Collecting data elements into segments (any of the DL/I LIST, DL/I MAP or DL/I REPORT command variants).

2. Identifying root segments for physical hierarchies (DL/I LIST command).

3. Defining the DL/I physical database hierarchies; resolving multiple physical parent conditions (DL/I MAP PHYSICAL HIERARCHY command).

4. Defining unidirectional logical relationships (DL/I MAP UNIDIRECTIONAL-RELTIONSHIPS command).

5. Ensuring that required access paths are defined (DL/I MAP ACCESS-PATHS command).

6. Identifying arrays and repeating groups (DL/I LIST and DL/I MAP ACCESS-PATHS commands).

7. Defining required secondary key access paths (DL/I REPORT SECONDARY-ACCESS-PATHS command).

8. Identifying sequence keys for the segments (DL/I REPORT SEGMENTS command).

9. Refining the DL/I database design.

Collecting Data Elements Into Segments

This process is performed automatically when you take these actions:

Step 1. Use the DesignManager MERGE and DESIGN commands to generate a normalized network schema in the Workbench Design Area from input USERVIEW and ENTITY members of the Modeling Dictionary

Step 2. Name the records. You can do this using the DesignManager LIST and NAME commands.

Step 3. Convert all the records into segments by issuing any variant of the DesignManager commands DL/I LIST, DL/I MAP, or DL/I REPORT. The names and numbers of the records are used as segment names and numbers. The data elements contained in each segment are the same as those comprising the corresponding record. The key of the record becomes the concatenated key of the segment.

Identifying Root Segments for Physical Hierarchies

DesignManager automatically identifies root segments for use as the starting segments of DL/I physical hierarchies. This is done when the records of a generated network schema are converted into segments. Conversion of records into segments has already been discussed.

A *root segment* is defined as one that has no physical parent. You can get a list of the root segments by entering this command:

```
DL/I LIST ROOTS SEGMENTS;
```

Defining the DL/I Physical Database Hierarchies; Resolving Multiple Physical Parent Conditions

Once the data elements have been collected into segments, the next major step in DL/I database design is to structure the segments into physical hierarchies. DesignManager does this for you automatically and displays the hierarchies when you enter this command:

```
DL/I MAP PHYSICAL HIERARCHY;
```

A separate physical hierarchy is produced for each root segment identified, whether or not it has any physical dependent segments.

To summarize, the command displays each physical hierarchy defined by DesignManager. Each hierarchy begins on a new page with a root segment as the starting segment. A segment with more than one physical parent is displayed only once as a box. On each subsequent occurrence as a physical child segment, it is displayed as a pointer to the box.

Using the physical hierarchies displayed, it is now up to you, the database designer, to make the final design decisions required for the definition of the physical database hierarchies. It is likely that you will define a separate physical database from each physical hierarchy produced and displayed in the DL/I MAP PHYSICAL HIERARCHY output. Each hierarchy displayed in the MAP output as a root segment with no dependent segments is a candidate for a root-only physical database.

For each physical database defined (other than single segment databases), you must determine the best *left-to-right* order for segments dependent on the same physical parent, as described later.

Having defined the physical database hierarchies, you must resolve any inconsistencies highlighted in the command output indicating segments which are directly dependent on two or more physical parents, as illustrated in this diagram:

```
+---------------+              +--------------+
| PARENT-1      |              | PARENT-2     |
|               |              |              |
+---------------+              +--------------+
         |                              |
         +------------------------------+
                        |
                +---------------+
                | CHILD-A       |
                |               |
                +---------------+
```

There are a number of methods you can use to solve such problems, as described in the following paragraphs.

First, you can choose to define to DL/I a physical duplicate of the segment under each of its physical parents; that is, as a separate physical child segment. When such duplication occurs in the same physical database, you must be sure to give a different name to each of the segment's duplicates.

A second method of resolving such a problem is to define to DL/I one or more *bidirectional logical relationships*. If the segment has two physical parents, only one bidirectional relationship need be defined. If the segment has three physical parents, two bidirectional logical relationships are required, and so forth.

To be able to specify bidirectional logical relationships, you will need to introduce another DL/I segment type, the *logical child segment*. This method can be implemented in two ways, as indicated in the next two diagrams, which are based on the inconsistency illustrated in the diagram shown on the last screen.

If only a single bidirectional logical relationship is required, you can choose to define the physical child segment itself as a logical child segment and pair it with a new logical child segment. Each will be a physical child of one of the physical parents and a logical child of the other, as shown in this diagram:

```
+---------------+                +---------------+
| PARENT-1      |<----------+    | PARENT-2      |
|               |    +------|--->|               |
+---------------+    |      |    +---------------+
        |            |      |            |
        |            |      |            |
+---------------+    |      |    +---------------+
| CHILD-A       |----+      |    |PAIRED-LOGICAL-|
|               |           +----|  CHILD-A      |
+---------------+                +---------------+
```

Alternatively, you can pair two new logical child segments, where:

- One logical child is defined to be a direct dependent of the physical child segment (which itself remains a child of one of the physical parents).
- The second logical child (the paired segment) is a physical child of the second physical parent.

This is illustrated in this diagram:

```
+---------------+               +---------------+
| PARENT-1      |               | PARENT-2      |
|               |    +--------->|               |
+---------------+    |          +---------------+
        |            |                  |
        |            |                  |
+---------------+    |                  |
| CHILD-A       |<---|-----+            |
|               |    |     |            |
+---------------+    |     |            |
        |            |     |            |
        |            |     |            |
+---------------+    |     |    +---------------+
| LOGICAL-      |----+     |    |PAIRED-LOGICAL-|
| CHILD-A       |          +----|  CHILD-A      |
+---------------+               +---------------+
```

You must choose the second way of defining a bidirectional logical relationship, that of specifying two new logical child segments, if either of these conditions exist:

- If you need to define more than one bidirectional logical relationship due to the physical child segment having more than two physical parents.
- If there needs to be more than one logical relationship specified along the same hierarchical path, due to a physical child segment and its physical parent both having more than one physical parent. This is because a logical child segment cannot be a direct dependent of another logical child segment.

One other factor that should be considered is that if the logical relationship will rarely be crossed, the second way of specifying logical children is more efficient, because DL/I will not then be called upon to maintain the pointers.

Defining Unidirectional Logical Relationships

Once you have your physical database hierarchies specified, with the appropriate bidirectional logical relationships defined, the next major step in the database design process is to identity the inherent DL/I *unidirectional logical relationships*. DesignManager does this for you automatically and displays the unidirectional relationships for you when you enter this command:

```
DL/I MAP UNIDIRECTIONAL-RELATIONSHIPS;
```

Each relationship identified is from a logical child segment to its logical parent segment, where the former contains the concatenated key of the latter.

To summarize here, the command outputs the unidirectional logical relationships in the form of one or more diagrams. Each diagram begins on a new page with a starting logical child segment. A logical parent segment which is pointed to by more than one logical child segment is displayed only once as a box. Each of its subsequent occurrences as a logical parent segment is highlighted by being displayed as a pointer to the box.

Using the displayed output, it is now up to you to make the final design decisions for defining the unidirectional logical relationships to DL/I. Consider, for example, the case of a diagram that contains only a single unidirectional logical relationship (one logical child segment pointing to its logical parent segment), as shown in the following diagram:

```
+-------------+
| PHYSICAL-   |
| PARENT      |
+-------------+
     |
+-------------+                      +-------------+
| SEG-A       |--------------------->| SEG-B       |
|             |                      |             |
+-------------+                      +-------------+
```

You can define such a relationship to DL/I in either of two ways.

First of all, you can choose to define the relationship just as it is displayed in the output diagram; that is, you can define the displayed logical child segment itself directly as the logical child pointing to the logical parent, as shown in the previous diagram.

Alternatively, you can create a logical child segment as a physical dependent of the displayed segment, defining it as the logical child pointing to the logical parent, as shown in this diagram:

```
+--------------+                       +--------------+
|PHYSICAL-     |             +--------->| SEG-B        |
|PARENT        |             |          |              |
+--------------+             |          +--------------+
       |                     |
+--------------+             |
| SEG-A        |             |
|              |             |
+--------------+             |
       |                     |
+--------------+             |
| LOGICAL-     |---------------+
| CHILD        |
+--------------+
```

There are a number of circumstances requiring use of the latter method, that of creating a new logical child segment as a direct physical dependent of the displayed segment. In particular, you must use the latter method in any of the following circumstances:

- The displayed segment is a root segment of a physical database.
- It points to more than one logical parent (it is the logical child in several logical relationships).
- It has already been defined to DL/I as a logical child in another logical relationship (in which case you must also redefine the previous logical relationship by specifying a new logical child which is a direct dependent of the displayed logical child).
- It is a physical child of another logical child segment.
- It is also a logical parent segment in another logical relationship.

Taking the last instance listed above as an illustration, suppose that a segment is displayed as the logical parent segment in one logical relationship, and, in turn, as the logical child segment in a second logical relationship, as shown in this diagram:

```
+---------------+
| PHYSICAL-     |
| PARENT        |
+---------------+
       |
+---------------+      +--------------+      +-------------+
| SEG-A         |----->| SEG-B        |----->| SEG-C       |
|               |      |              |      |             |
+---------------+      +--------------+      +-------------+
```

In this case, you must define a new logical child segment as a physical child of the displayed segment, as shown in this diagram:

```
+--------------+      +---------------+      +--------------+
| PHYSICAL-    |  +-->| SEG-B         |  +-->| SEG-C        |
| PARENT       |  |   |               |  |   |              |
+--------------+  |   +---------------+  |   +--------------+
       |          |           |          |
+--------------+  |   +---------------+  |
| SEG-A        |--+   | LOGICAL-      |--+
|              |      | CHILD         |
+--------------+      +---------------+
```

As a consequence, before you define a logical parent segment, you should first check to see if it has already been defined as a logical child segment in a previous logical relationship. If it has, then you must redefine the previous relationship by specifying a new logical child segment as a dependent of the logical parent segment.

As a final example, you may find that a logical parent segment is displayed which points back to the logical child segment, because the logical parent contains the concatenated key of the logical child. In this case, the two unidirectional logical relationships can be defined to DL/I as a single bidirectional relationship.

Ensuring that Required Access Paths are Defined

Once you have defined the physical hierarchies to DL/I together with any required unidirectional and bidirectional logical relationships, then you must ensure that all *primary key access paths* derived from the input userviews and entities are catered for. Any such access requirements which have not already been satisfied implicitly (via the previously defined logical relationships), you must now define explicitly.

DesignManager automatically displays all derived primary key access paths for you when you enter this command:

```
DL/I NAP ACCESS-PATHS;
```

To summarize, the command outputs the access paths in the form of one or more diagrams. Each diagram begins on a new page with a starting segment. A destination segment which is pointed to by more than one segment is displayed only once as a box. Each of its subsequent occurrences as a destination segment is highlighted by being displayed as a pointer to the box.

Using the displayed output, it is now up to you to ensure that all the access paths displayed in the diagram are indeed defined to DL/I. Once you have studied the output, you may find that some of the access path requirements have already been satisfied implicitly.

Suppose, for example, that your earlier output from the DL/I MAP PHYSICAL HIERARCHY command indicated a physical child segment with more than one physical parent, as shown in the following diagram:

```
 +--------------+              +--------------+
 | PARENT-1     |              | PARENT-2     |
 |              |              |              |
 +--------------+              +--------------+
         |                              |
         +------------------------------+
                        |
                +--------------+
                | CHILD-A      |
                |              |
                +--------------+
```

In this case, you would already have defined a bidirectional logical relationship to DL/I, as illustrated in the following diagram:

```
+--------------+               +---------------+
| PARENT-1     |<---------+    | PARENT-2      |
|              |    +-----|--->|               |
+--------------+    |     |    +---------------+
        |           |     |             |
        |           |     |             |
+--------------+    |     |    +---------------+
| CHILD-A      |----+     |    |PAIRED-LOGICAL-|
|              |          +----|   CHILD-A     |
+--------------+               +---------------+
```

Therefore, if the output of the DL/I MAP ACCESS-PATHS command shows that an access path is required from the PARENT-1 segment to the PARENT-2 segment, as illustrated in the following diagram, this access requirement would automatically be satisfied by the previously defined bidirectional relationship:

```
+--------------+              +--------------+
| PARENT-1     |------------>>| PARENT-2     |
|              |              |              |
+--------------+              +--------------+
```

Similarly, suppose your earlier output from the DL/I MAP UNIDIRECTIONAL-RELATIONSHIPS command indicated that the physical child of one segment is the logical child of another, as shown in the next diagram:

```
+--------------+
| PHYSICAL-    |
| PARENT       |
+--------------+
        |
        |
+--------------+               +---------------+
| LOGICAL-     |-------------->| LOGICAL-      |
| CHILD        |               | PARENT        |
+--------------+               +---------------+
```

In this case, you would already have defined a unidirectional logical relationship to DL/I from the logical child to the logical parent. Therefore, if the output of the DL/I MAP ACCESS-PATHS command shows that an access path is required from the PHYSICAL-PARENT segment to the LOGICAL-PARENT segment, as illustrated in the following diagram, this access requirement would automatically be satisfied by the previously defined unidirectional relationship:

```
+--------------+               +---------------+
|PHYSICAL-     |------------->>| LOGICAL-      |
|PARENT        |               | PARENT        |
+--------------+               +---------------+
```

Access paths requirements which are implicitly satisfied by previously defined logical relationships, as illustrated in the above examples, need not be redefined to DL/I. However, if your output indicates access path requirements which are not already satisfied implicitly, then you must define these paths to DL/I explicitly.

If, for example, a simple access path is displayed, that is, a diagram with only two segments appearing on the page and in one direction only, you can satisfy the access requirement by defining to DL/I a unidirectional logical relationship, where the segment requiring access is defined as the logical child and the destination segment is defined as the logical parent.

If access paths are displayed (in one direction only) from a single segment to two or more destination segments, you can satisfy the requirement by defining several unidirectional logical relationships to DL/I. For each such relationship, you must define a new logical child segment which is a physical child segment of the segment from which access is required and which has one of the destination segments as its logical parent.

If an access path is displayed between two segments in both directions, you can satisfy the access requirement by defining a bidirectional logical relationship to DL/I. In this case, two new logical child segments must be defined, where each is a physical child of one of the two segments displayed, and where each of the segments displayed is a logical parent of one of the logical child segments.

As a final example, suppose that an access path is required from one segment to a second segment, which in turn requires access to a third segment, as shown in the following diagram:

```
+--------------+      +--------------+      +--------------+
| SEG-A        |---->>| SEG-B        |---->>| SEG-C        |
|              |      |              |      |              |
+--------------+      +--------------+      +--------------+
```

In this case, two new logical child segments must be created which are, respectively:

- A physical child of the first segment requiring access, SEG-A, and a logical child of the first destination segment, SEG-B.
- A physical child of the second segment requiring access, SEG-B, and a logical child of the second destination segment, SEG-C.

This is illustrated in the following diagram:

```
+--------------+      +--------------+      +-------------+
| SEG-A        |  +-->| SEG-B        |  +-->| SEG-C       |
|              |  |   |              |  |   |             |
+--------------+  |   +--------------+  |   +-------------+
      |           |                     |
      |           |                     |
+--------------+  |   +--------------+  |
| LOGICAL-     |--+   | LOGICAL-     |--+
| CHILD-1      |      | CHILD-2      |
+--------------+      +--------------+
```

Identifying Arrays and Repeating Groups

You have now defined to DL/I your physical database hierarchies plus all their required logical relationships (except perhaps for one or more secondary key access paths which you may find are not eligible for definition to DL/I as secondary indexes).

If you have defined any root-only databases, then one or more of the corresponding root segments may qualify as an *array* or *repeating group* and can, if you choose, be defined to DL/I in another way.

A root segment of a root-only database can be identified as an array or a repeating group if both of the following (primary key) access path conditions are satisfied:

- There is one and only one access path specified to the root segment
- There is no access path specified from the root segment to any other segment.

In these circumstances, the root segment may be considered to be an array or repeating group of the segment requiring access. In this event, you can either leave it as a root segment of a root-only database or you can define it to DL/I in either of these ways:

- You can merge the root segment with the segment requiring access.
- You can define the root segment as a physical child of the segment requiring access.

Consider, for example, the following illustration of an employee and his or her dependents. Suppose you had input these functional and multivalued dependencies (FDs and MVDs):

```
EMPLOYEE-NUMBER -----> EMPLOYEE-DETAILS
EMPLOYEE-NUMBER ---->> DEPENDENT-NAME
```

where EMPLOYEE-DETAILS includes the pertinent data elements that are functionally dependent on EMPLOYEE-NUMBER.

If you named the resulting records (in the generated network schema) EMPLOYEE and DEPENDENT, a subsequent DL/I MAP ACCESS- PATHS command would show the following access path:

```
+--------------+                +---------------+
| EMPLOYEE     |-------------->>| DEPENDENT     |
|              |                |               |
+--------------+                +---------------+
```

where the destination segment, DEPENDENT, is also the root segment of a root-only database. The DEPENDENT segment thus satisfies the conditions given above for an array or a repeating group. As such, you can leave it as the root segment of the root-only database, or you can define it to DL/I in either of the following ways:

- You can define the DEPENDENT segment as a physical child of the EMPLOYEE segment, as illustrated in this diagram:

```
+--------------+
| EMPLOYEE     |
|              |
+--------------+
        |
        |
+--------------+
| DEPENDENT    |
|              |
+--------------+
```

- Alternatively, you can merge the DEPENDENT segment as part of the EMPLOYEE segment, so that the attribute, DEPENDENT-NAME, of the DEPENDENT segment becomes instead an attribute of the EMPLOYEE segment, where it is multiply-determined by the key EMPLOYEE-NUMBER. This is illustrated in the following diagram of the resulting EMPLOYEE segment:

```
           +----------++---(F)----+---(M)----+
EMPLOYEE   |EMPLOYEE- ||EMPLOYEE- |DEPENDENT-|
           |NUMBER    ||DETAILS   |NAME      |
           +----------++----------+----------+
```

where (F) indicates one or more functionally-determined attributes and (M) indicates one or more multiply-determined attributes.

Defining Required Secondary Key Access Paths

There is still one major task remaining in your DL/I database design. You must identify all access paths which are required via secondary search keys and define them to DL/I. Some secondary key access paths can be defined to DL/I by means of secondary indexes and others require the specification of logical relationships.

DesignManager identifies the required secondary key access paths for you when you enter this command:

```
DL/I REPORT SECONDARY-ACCESS-PATHS;
```

Secondary key access paths are derived by DesignManager from your input userviews.

To summarize, each secondary key access path is defined from a non-key set of data elements in a source segment to the key of a target segment, where the source segment can itself be the target segment. Each line of output describes a distinct secondary key access path, and lists, by number and name (if assigned), the source segment, the target segment, and the non-key set of data elements comprising the secondary search key.

The only secondary key access paths which are eligible for definition to DL/I as secondary indexes are those in which either:

- The source segment is a dependent segment, either direct or indirect, of the target segment.
- The source segment is itself the target segment.

In such a case, you can define the index source segment, the index target segment, and the index pointer segment from the report output.

Suppose, for example, this physical hierarchy has been displayed in the output of the DL/I MAP PHYSICAL HIERARCHY command:

```
+--------------+
|  SEG-A       |
|              |
+--------------+
        |
        |
+--------------+
|  SEG-B       |
|              |
+--------------+
        |
        |
+--------------+
|  SEG-C       |
|              |
+--------------+
```

Suppose further that a secondary key access path is required from a non-key set of data elements in SEG-C (the source segment) to the key of SEG-A (the target segment), as indicated in the following diagram:

```
+--------------+             +--------------+
|  SEG-C       |------------>>|   SEG-A      |
|              |             |              |
+--------------+             +--------------+
```

Because SEG-C is a dependent segment of SEG-A, you can define this path to DL/I as a secondary index, as shown in the diagram below:

```
+--------------+    target segment
| SEG-A        |<------------------------+
|              |                         |
+--------------+                         |
      |                                  |
      |                                  |
+--------------+                         |
|  SEG-B       |                         |
|              |                         |
+--------------+                         |
      |                                  |
      |                                  |
+--------------+  source segment +--------------+
|  SEG-C       |---------------->|INDEX POINTER |
|              |                 |SEGMENT       |
+--------------+                 +--------------+
```

If a secondary key access path is reported which does not satisfy the above conditions for a DL/I secondary index, then the only way you can define it to DL/I is by means of a logical relationship. As in the case of primary key access paths, you may already have defined a unidirectional or bidirectional logical relationship which satisfies the access requirement implicitly.

Otherwise, you must define it explicitly to DL/I. The same guidelines apply that were given for defining primary key access paths by means of logical relationships (except that, if more than one access path is required between the same source and target segments, then a single logical relationship will satisfy the requirements of them all).

Identifying Sequence Keys for the Segments

Now that you have defined to DL/I your physical databases plus required logical relationships and secondary indexes, you still must identity the sequence key for each segment. You can do this by entering this command:

```
DL/I REPORT SEGMENTS;
```

This command produces a report on all the segments generated, including for each its name (if one has been assigned) and number, its concatenated key, and all the non-key fields.

You can determine the sequence key for any non-root segment by removing, from its listed concatenated key, the concatenated key of its physical parent (that is, by removing the sequence key of each segment in the hierarchical path leading from the root segment to the segment being processed). The sequence key for a root segment is just the concatenated key reported for the segment in the command output.

Refining the DL/I Database Design

Although you now have a sound structure for your DL/I databases and have identified the sequence key of each segment, there still remains the task of deciding, for each physical parent segment, what is the best left to right order for its direct dependents. In addition, if you have defined any virtually paired logical relationships, you must decide where to place the real logical child and where to place the virtual logical child.

In some instances these decisions may depend upon a number of factors:

- The anticipated volume of data
- Expected access frequencies
- Desired response times.

You can, for example, decide to place large volume data at the end of a database or to place it in a separate database of its own.

Where access frequencies and/or response times are important in making your decisions, DesignManager can help significantly. If you have the optional Load Factor Calculation facility (selectable unit DSR-PH1O) installed, you can compare the appropriate load factors calculated for the data elements comprising the sequence keys of the segments to assist you in making your decisions.

If response time is important, you could place segments whose sequence keys have faster response times to the left of segments whose sequence keys have slower response times.

Where access frequencies are important, segments whose sequence keys have higher access frequencies could be placed to the left of segments whose sequence keys have lower access frequencies.

If a virtually paired logical relationship has been defined, where neither of the logical child segments has any dependent segments, you might wish to select the segment with the higher access frequency as the real logical child.

Index